Working in the View wth Helpers and Layouts

SI539 - Chapter 6 Lenz

Textbook: Build Your own Ruby on Rails Application by Patrick Lenz (ISBN:978-0-975-8419-5-2)

View Layer Techniques

- Flash - Message storage for one "view"
- Helpers in Embedded Ruby
- RHTML Templates to "wrap" views
- RHTML Includes - Render partial
- Building your own helpers
- Using Redirect in the controller
- Form Helpers
- Scaffolding

Flash

- Flash is a short-term storage area that is cleared after each view.
- It is a place to put messages that you want to appear once
- Usually it is some type of feedback like "New record added" or "Delete failed because of database error"
- Sometimes these are errors and sometimes they are feedback

```
  def new
    @story = Story.new(params[:story])
    if request.post?
      @story.save
      flash[:notice] = 'Story submission succeeded'
      redirect_to :action => 'index'
    end
  end
```

Flash in Action! *

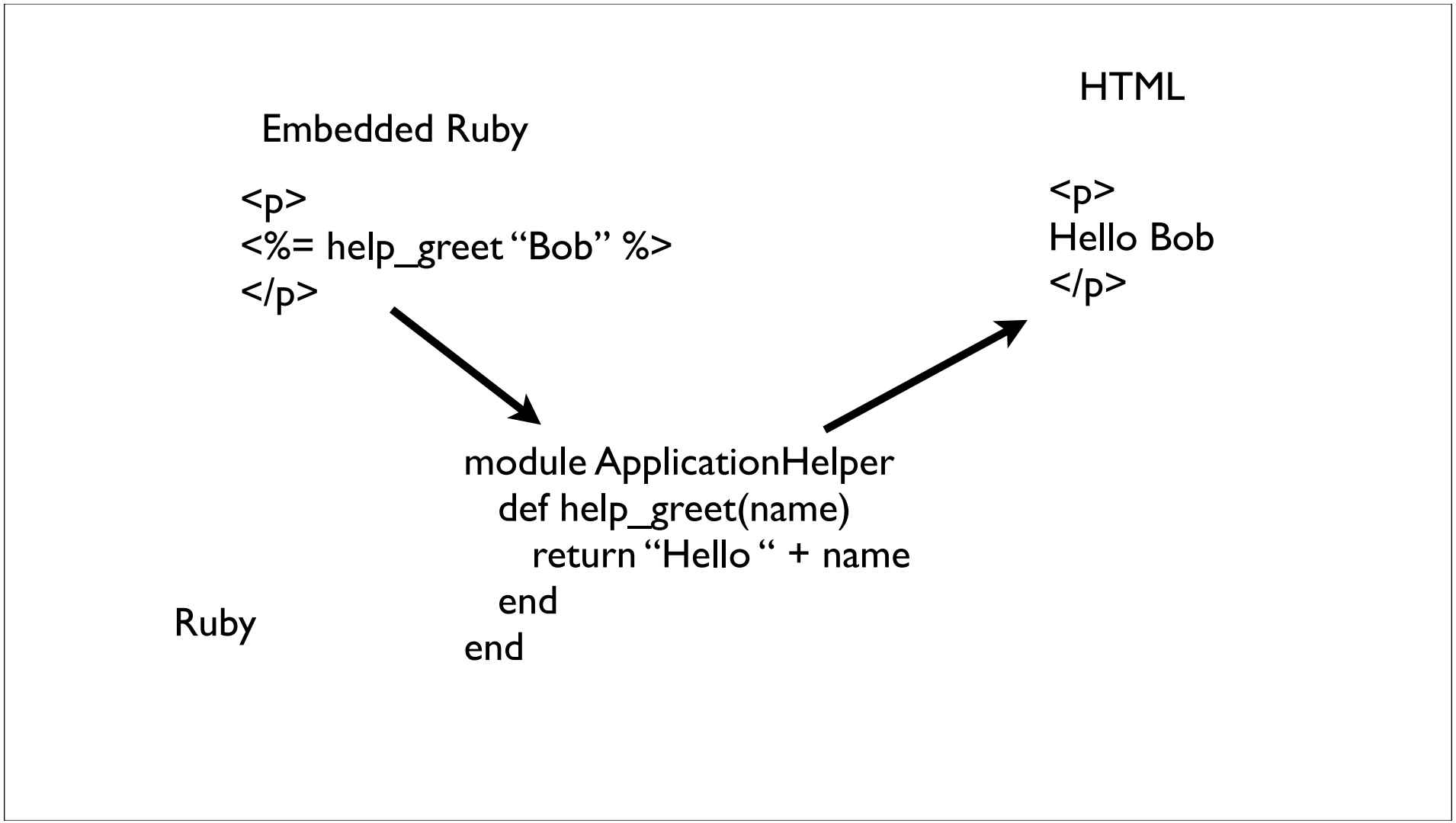
```
<div id="content">
  <h1>Shovell</h1>
   <% unless flash[:notice].blank? %>
     <div id="notification"><%= flash[:notice] %></div>
   <% end %>
  <%= yield %>
</div>
```

* Actually it is in the action *and* the layout in this example

What is a Helper?

- A helper is a form of DRY - "Don't Repeat Yourself"
- A short-cut to producing simple repetitive HTML
- Reduces little mistakes like forgetting < or > or "
- Avoid "hard coding" things like the controller name or URL pattern
- An example of calling Ruby code from Embedded Ruby in a template

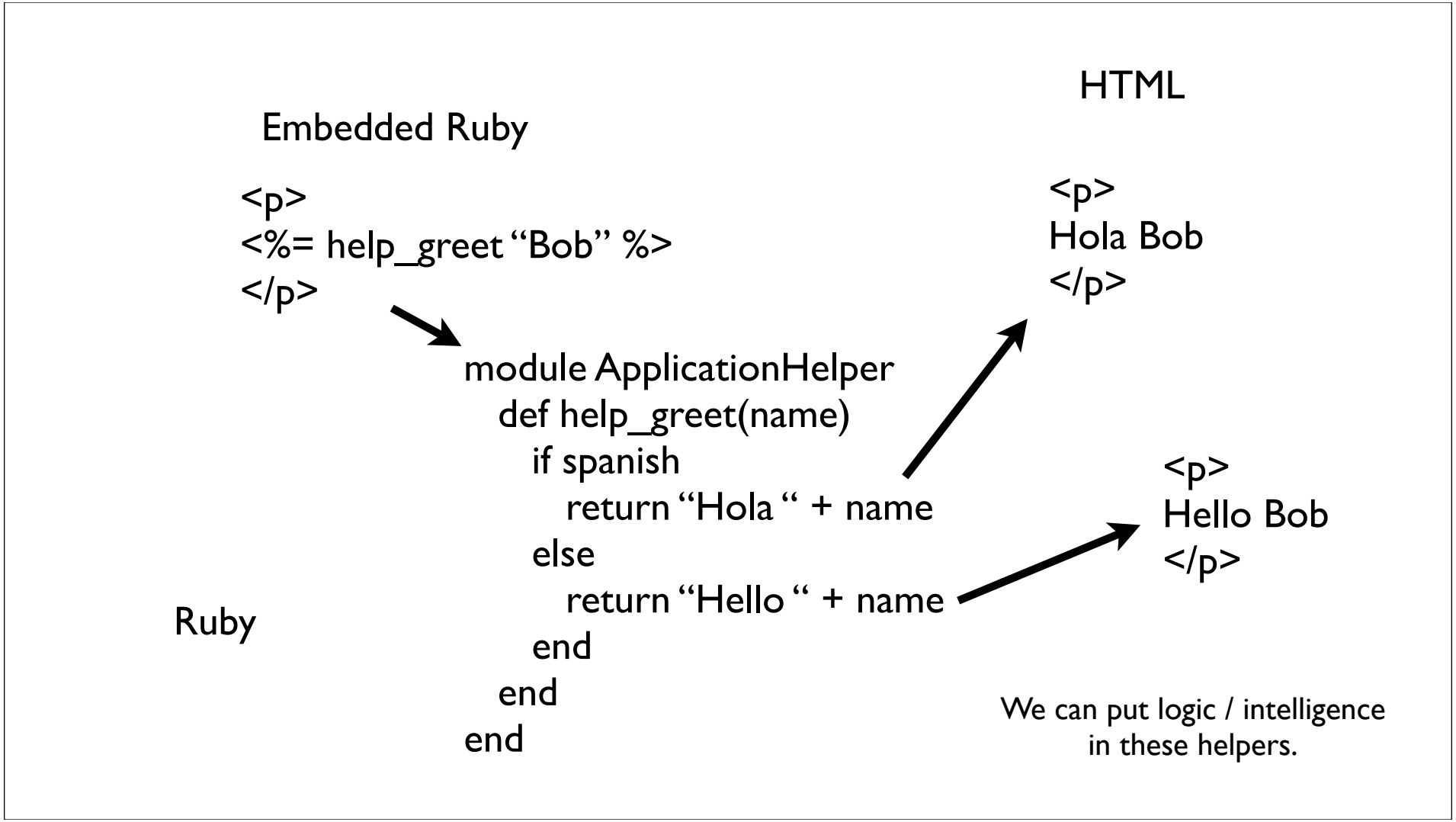
http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html


HTML
Embedded Ruby

<%= help_greet "Bob" %>

Hello Bob

module ApplicationHelper
def help_greet(name)
return "Hello " + name
end
end
Ruby


Embedded Ruby

<%= help_greet "Bob" %>

module ApplicationHelper
def help_greet(name)
if spanish
return "Hola " + name
else
return "Hello " + name
end
end
end
Ruby
HTML

Hola Bob

Hello Bob

We can put logic / intelligence
in these helpers.

url_for Helper

url_for :action => "index"

/One/index

<a href="<%= url_for :action => "index" %>">

<a href="/One/index">

This is a very simple example - others can be quite complex

stylesheet_link_tag Helper

```
<%= stylesheet_link_tag "style.css" %>
```

```
<link href="/stylesheets/style.css?1205532649" media="screen"
rel="Stylesheet" type="text/css" />
```

This is a very simple example - others can be quite complex

image_tag Helper

```
<%= image_tag "ruby-cap-100wide.jpg", :alt => "Ruby cap logo image" %>
```

```
<img alt="Ruby cap logo image" src="/images/ruby-cap-100wide.jpg?1205325643" />
```

This is a very simple example - others can be quite complex

link_to Helper

link_to "About", { :action => ‘index’ }

About

This is a very simple example - others can be quite complex

link_to Helper

link_to "Visit Other Site", "http://www.rubyonrails.org/"

Visit Other Site

This is a very simple example - others can be quite complex

```
<%= link_to ( image_tag("ruby-cap-100wide.jpg",
:alt => "Ruby cap logo image") , "http://www.rubyonrails.org/") %>
```

link_to(name, options = {}, html_options = nil, *parameters_for_method_reference)

Creates a link tag of the given `name` using a URL created by the set of `options`. See the valid options in the documentation for ActionController::Base#url_for. It's also possible to pass a string instead of an options hash to get a link tag that uses the value of the string as the href for the link. If nil is passed as a name, the link itself will become the name.

The `html_options` will accept a hash of html attributes for the link tag. It also accepts 3 modifiers that specialize the link behavior.

- `:confirm => 'question?'`: This will add a JavaScript confirm prompt with the question specified. If the user accepts, the link is processed normally, otherwise no action is taken.
- `:popup => true || array of window options`: This will force the link to open in a popup window. By passing true, a default browser window will be opened with the URL. You can also specify an array of options that are passed-thru to JavaScripts window.open method.
- `:method => symbol of HTTP verb`: This modifier will dynamically create an HTML form and immediately submit the form for processing using the HTTP verb specified. Useful for having links perform a POST operation in dangerous actions like deleting a record (which search bots can follow while spidering your site). Supported verbs are :post, :delete and :put. Note that if the user has JavaScript disabled, the request will fall back to using GET. If you are relying on the POST behavior, your should check for it in your controllers action by using the request objects methods for post?, delete? or put? .

You can mix and match the `html_options` with the exception of :popup and :method which will raise an ActionView::ActionViewError exception.

```
link_to "Visit Other Site", "http://www.rubyonrails.org/", :confirm => "Are you sure?"
link_to "Help", { :action => "help" }, :popup => true
link_to "View Image", { :action => "view" }, :popup => ['new_window_name', 'height=300,width=600']
link_to "Delete Image", { :action => "delete", :id => @image.id }, :confirm => "Are you sure?", :method => :delete
```

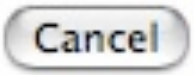
http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html

button_to

- Clever - works like link_to but makes a complete small form with an action and a single submit button
- Messes up layout when used within another form

```
<%=   button_to "Cancel", :action => "index" %>
```

```
<form method="post" action="/one"
class="button-to"><div><input type="submit"
value="Cancel" /></div></form>
```



Reusable HTML
Don't Repeat Yourself (D.R.Y.)

Two forms of Reusable HTML

- A template that is used to "wrap" all of the view files which includes title, head material, etc. - this is called a "template"
- A partial template that contains a fragment of HTML which is included several places
- Both are in RHTML and as such can include Embedded Ruby and have access to the class variables (variables that start with @)

Layout

- A layout is common HTML used across a controller's many views
- /app/views/layouts/stories.rhtml
  - stories = controller name
- /app/views/layouts/application.html
  - Wraps all views in the application
- Commonly used for header material, CSS, etc to keep from pasting this into each view separately - thus making a maintenance nightmare

app/views/layouts/application.rhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <title>Assignment 8 - SI539</title>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <%= stylesheet_link_tag "style.css" %>
</head>
<body>
   <div id="header">
     <h1>Welcome to SI539</h1>
   </div>
   <%= render :partial => 'nav' %>
   <div id="bodycontent">
      <%= yield %>
   </div>
</body>
</html>
```

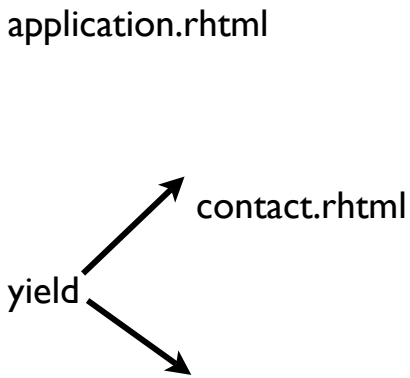
Sample Layout

The yield line includes the text of the actual view for this request. So HTML in this layout is "wrapped around" the view's output. Note the "=" yield "produces" the enclosing view's output.

app/views/one/contact.rhtml

```
<h2>Welcome to the SI539 Sample Site</h2>
<p>For more information, send mail to
  <a href="mailto:info@si539.com">info@si539.com</a>.</p>
```

application.rhtml wraps each of the view files. So there is no need to put the header material in the view files. The view files are inserted into the layout at the "yield" point. Normal view files can be qiute short and we have no repeated material from view to view.



Without Layouts

index.rhtml

contact.rhtml

pictures.rhtml

thanks.rhtml

With Layouts

application.rhtml

yield

index.rhtml

pictures.rhtml

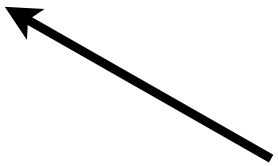
contact.rhtml

thanks.rhtml

app/views/layouts/application.rhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
   <title>Assignment 8 - SI539</title>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
   <%= stylesheet_link_tag "style.css" %>
</head>
<body>
   <div id="header">
     <h1>Welcome to SI539</h1>
   </div>
   <%= render :partial => 'nav' %>
   <div id="bodycontent">
      <%= yield %>
   </div>
</body>
</html>
```

Partial Render



The render helper with a parameter of :partial indicates to include a template from a file. Partial implies not to apply the application wrapper.

app/views/one/_nav.rhtml

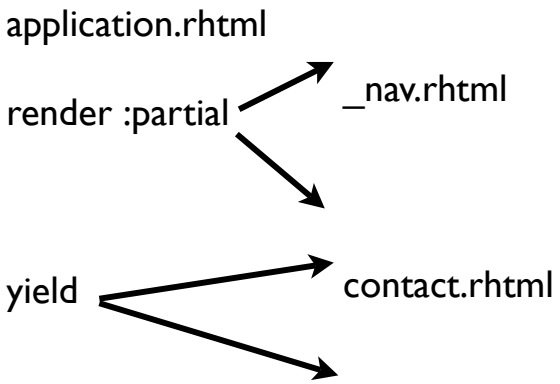
```
<div id="navigation">
  <ul>
    <li><a href="<%= url_for :action => "index" %>">About</a></li>
    <li><a href="<%= url_for :action => "contact" %>">Contact</a></li>
    <li><a href="<%= url_for :action => "pictures" %>">Pictures</a></li>
    <li><a href="<%= url_for :action => "members" %>">Membership</a></li>
    <li><a href="<%= url_for :action => "join" %>">Application</a></li>
  </ul>
</div>
```

Partial render views file names start with underscore. Tehy may include Embedded Ruby.

app/views/one/_nav.rhtml

```
<div id="navigation">
  <ul>
    <li><a href="<%= url_for :action => "index" %>">About</a></li>
    <li><a href="<%= url_for :action => "contact" %>">Contact</a></li>
    <li><a href="<%= url_for :action => "pictures" %>">Pictures</a></li>
    <li><a href="<%= url_for :action => "members" %>">Membership</a></li>
    <li><a href="<%= url_for :action => "join" %>">Application</a></li>
  </ul>
</div>
```

Render partial includes the HTML from the file after processing the ERB in the file.
Partial rendering can be done in any file - not just a template file.



Your Own Helpers

- You can make your own helpers that you can call from your templates in Embedded Ruby
- Write a method within one of the helper files
  - app\helpers\application.rb (For whole application)
  - app\helpers\one_controller.rb (For one controller)

Problem - Generate Navigation

```
<div id="navigation">
  <ul>
    <li><a href="<%= url_for :action => "index" %>"
                class=”selected”>About</a></li>
    <li><a href="<%= url_for :action => "contact" %>">Contact</a></li>
    <li><a href="<%= url_for :action => "pictures" %>">Pictures</a></li>
    <li><a href="<%= url_for :action => "members" %>">Membership</a></li>
    <li><a href="<%= url_for :action => "join" %>">Application</a></li>
  </ul>
</div>
```

Only one of the <li> entries is selected. It depends on which action we are in.

Helper - Generate Navigation

```
<div id="navigation">
  <ul>
    <%= do_nav_entry("About", "index") %>
    <%= do_nav_entry("Contact", "contact") %>
    <%= do_nav_entry("Pictures", "pictures") %>
    <%= do_nav_entry("Membership", "members") %>
    <%= do_nav_entry("Application", "join") %>
  </ul>
</div>
```

We call our own helper which will generate the entire <li> tag with the href, selected, and everything. We pass two parameters - the text ti display and the action for this text.

Observation...

```
Processing OneController#members (for 127.0.0.1 at 2008-03-18 23:42:30) [GET]
  Session ID: 0bd5ccb2a76b7b6a1d03beed81662402
  Parameters: {"action"=>"members", "controller"=>"one"}
```

On every request, the name of the current action is in the parameters map under the key “action”.

```
module OneHelper
  def do_nav_entry(textparam, actionparam)
    logger.info "In do_nav_entry text=#{textparam} action=#{actionparam}"

    seltext = ''
    if actionparam == params[:action]
      logger.info "We found the matching action!"
      seltext = ' class="selected"'
    end

    # call the url_for helper get the URL for our action
    urltext = url_for(:action => actionparam)
    logger.info "urltext=#{urltext}"
    return '<li><a href="' + urltext + '"'+ seltext+'>'+textparam+"</a></li>"
  end
end
```

```
Processing OneController#pictures (for 127.0.0.1 at 2008-03-18 23:45:00)
  Session ID: 0bd5ccb2a76b7b6a1d03beed81662402
  Parameters: {"action"=>"pictures", "controller"=>"one"}
Rendering  within layouts/application
Rendering one/pictures
In do_nav_entry text=About action=index
urltext=/one
In do_nav_entry text=Contact action=contact
urltext=/one/contact
In do_nav_entry text=Pictures action=pictures
We found the matching action!
urltext=/one/pictures
In do_nav_entry text=Membership action=members
urltext=/one/members
In do_nav_entry text=Application action=join
urltext=/one/join
Rendered one/_nav (0.00431)
```

```
<div id="navigation">
  <ul>
    <%= do_nav_entry("About", "index") %>
    <%= do_nav_entry("Contact", "contact") %>
    <%= do_nav_entry("Pictures", "pictures") %>
    <%= do_nav_entry("Membership", "members
    <%= do_nav_entry("Application", "join") %>
  </ul>
</div>
```

Summary

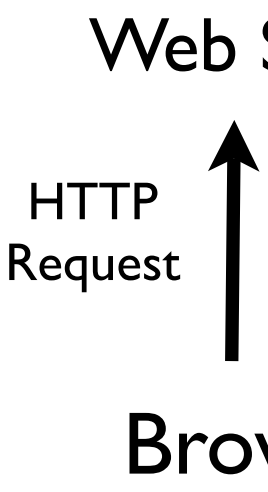
- Helpers and Layouts are a form of D.R.Y.
- They keep us from writing brittle HTML over and over
- The form_for helper writes HTML for us which produces a a hash map from request POST data suitable for use with a new operation for our underlying object
- The form_for helper encapsulates parameter naming and parameter parsing code - simplifying our life

In the Controller...

Additional Capabilities

- In a Rails ApplicationController you are provided the Request object to allow you to make decisions on the kind of request that you have
- Also you can send a special response called a "redirect" to the browser to tell the browser to "Go somewhere else"

Passing Parameters to The Server




```
GET /simpleform.html?yourname=fred
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
```

```
POST /simpleform.html
Accept: www/source
Accept: text/html
User-Agent: Lynx/2.4 libwww/2.14
Content-type: application/x-www-form-urlencoded
Content-length: 13
yourname=fred
```

```
<input type="text" name="yourname" id="yourid" />
```

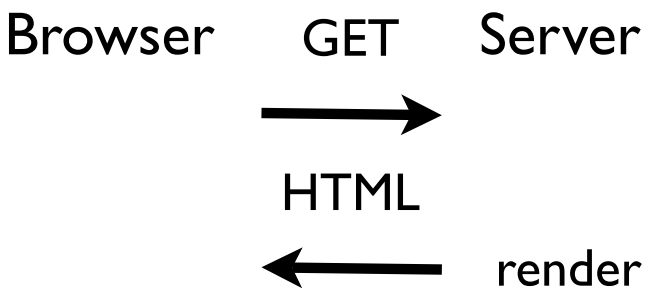
Request Object

```
if request.post?
  @story.save
  redirect_to :action => 'index'
end
```

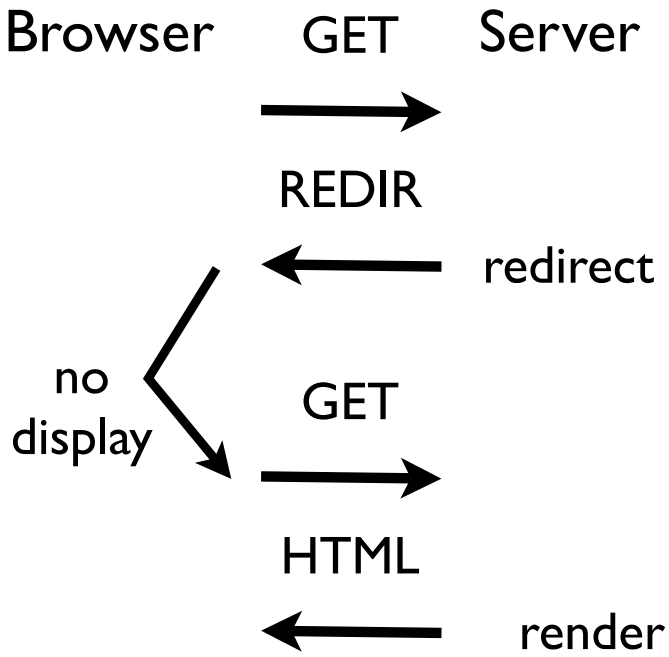
- When a browser requests a new page from the web server there are four types of request (part of HTTP protocol)
  - GET - Read/retrieve an object or page
  - POST - Update an object (also used for create)
  - PUT - Create a new object
  - DELETE - Destroy an object
- GET and POST are commonly used to submit form data

http://api.rubyonrails.com/classes/ActionController/AbstractRequest.html

Normal Request/Response



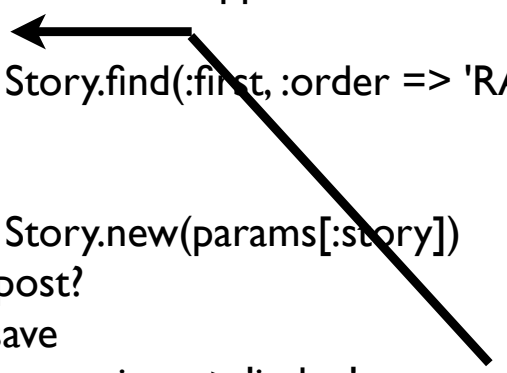
Redirect to new page



redirect_to in code

- Sometimes when we are done with one controller action we want to switch to another controller action to complete the request
- Usually this also switches views

```
class StoryController < ApplicationController
  def index
    @story = Story.find(:first, :order => 'RAND()')
  end
  def new
    @story = Story.new(params[:story])
    if request.post?
      @story.save
      redirect_to :action => 'index'
    end
  end
end
```



http://api.rubyonrails.com/classes/ActionController/Base.html

Scaffolding (quick)

Scaffolding

- Rails can generate a complete C.R.U.D. Application
  - Create, Read, Update, Delete
- Don't assume that scaffolding is the way to start each new project - this is seldom the case.
- Think of this as sample code to learn from and understand
- Good way to see Rails features in use

Scaffolding Steps

- Make a model, set up migration, generate scaffold controllers and views

```
ruby script/generate model Story

edit Migration file

rake db:migrate
ruby script/generate scaffold Story
```

Summary

- We have many capabilities in the name of "Don't repeat yourself"
- They may seem like a bunch of extra work - but even with a moderate sized web application / site - it is very nice to make changes one place and have them percolate throughout the site.